Zápočtový program

Implementace AVL-stromu

David Novák

ZS 2017-2018 3. roč. BBI, PřF UK

Programování I, NPRG030 Free Pascal 3.0.4

Anotace

Program implementuje datovou strukturu AVL-strom a umožňuje vložení a odstranění 32-bitových celých čísel, jakož i nalezení cest od kořene stromu až ke hledané uložené hodnotě. Hledanou hodnotou může být zadané číslo, minimum nebo maximum (z hodnot ve stromě uložených).

Algoritmus

AVL-strom je samovyvažující se binární vyhledávací strom. Binární trom je orientovaný graf složený z uzlů. Každý z uzlů má 0 až 2 syny a právě jednoho předka (s výjimkou vrcholu-kořene). Každý uzel má klíč (tedy číselný údaj, jejž do stromu ukládáme). Pro libovolný uzel A platí:

- má-li A levého syna, klíč levého syna < klíč A,
- má-li A levého syna, klíč levého syna > klíč A.

Samovyvažující se strom je strom takový, že při vkládání a odstraňování nových uzlů udržuje svou celkovou hloubku (maximální počet uzlů na cestě při jednosměrném průchodu od kořene až k uzlu bez synů) na hodnotě, která není za daných okolností zbytečně vysoká. Toho docilujeme pomocí úpravy pozic uzlů v rámci nevyváženého podstromu. Podstrom je nevyvážený, je-li levý podstrom podstatně hlubší než pravý, nebo naopak.

Pro každý uzel A v rámci AVL-stromu zavádíme rovnovážný faktor. Rovnovážný faktor je rozdílem hloubek pravého a levého podstromu. Je-li rovnovážný faktor uzlu A roven -2 nebo 2, je podstrom s kořenem A nevyvážený a je třeba jej vyvážit pomocí rotací. Má-li rovnovážný faktor hodnotu z {-1, 0, 1}, považujme zmíněný podstrom za vyvážený a nevyvažujme jej. Jiných hodnot rovnovážný faktor nabývat nemůže. Zaveďme následující pracovní pojmy.

- Je-li rovnovážný faktor uzlu A = -2, je (pod)strom s kořenem A *nalevo nevyvážený*;
- je-li rovnovážný faktor uzlu A = -1, (pod)strom s kořenem A *tíhne doleva*;
- je-li rovnovážný faktor uzlu A = 0, je (pod)strom s kořenem A *dokonale vyvážený*;
- je-li rovnovážný faktor uzlu A = 1, (pod)strom s kořenem A *tíhne doprava*;
- je-li rovnovážný faktor uzlu A = 2, je (pod)strom s kořenem A *napravo nevyvážený*.

Pro eliminaci nevyvážených stavů v případě potřeby provádíme na (pod)stromech rotace.

Rotace dělíme na

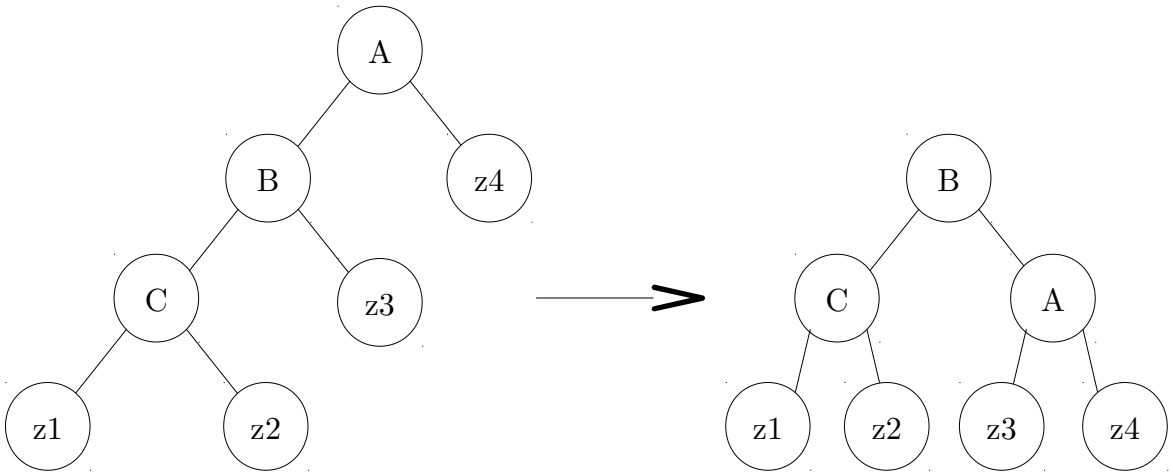
- jednoduché,
- dvojité.

Je-li (pod)strom s kořenem A nalevo nevyvážený a jeho levý syn (B) tíhne doleva, nazývejme toto případem *levo-levo*. Provádíme *pravou rotaci*, která vede ke vzniku (pod)stromu s kořenem B. Rovnovážný faktor nově vzniklého (pod)stromu je stejný, jako byl před krokem (vložení nebo odstranění uzlu), jenž nevyvážený stav vyvolal.

Je-li (pod)strom s kořenem A nalevo nevyvážený a jeho pravý syn (B) tíhne doleva, nazývejme toto případem *levo-pravo*. Provádíme *levou rotaci* podstromu s kořenem v B, následně *pravou* rotaci (pod)stromu s kořenenem v A (tj. dvojitou rotaci levo-pravou). Kořenem nového (pod)stromu se po těchto dvou rotací stává uzel, jenž byl na začátku uzlu B pravým synem.

Analogicky přistupujeme k situacím *pravo-pravo* a *pravo-levo*, rotace jsou vůči předchozím dvěma případům symetrické.

Rotace může být znázorněna schematicky takto:



z1 až *z4* jsou jednotlivé uzly nebo podstromy. Znázorněna je zde rotace pravá.

Operace vkládání a odstraňování uzlů a průchod stromem mají časovou složitost O(log *n*). Časová složitost rotací je konstatní. AVL-stromy jsou pricipielně podobné stromům červeno-černým. Oproti červeno-černým stromům mohou být AVL-stromy rychlejší v případech, kdy potřebujeme často stromem procházet[(1)]. Mechanismus, kterým je strom vyvažován, má totiž přísnější kritéria.

(1) *https://web.stanford.edu/~blp/papers/libavl.pdf*

Volba zadání

Téma AVL-stromů jsem si vybral, jelikož jsem zprvu nechápal postup při rotacích a chtěl jsem se jím dále zabývat.

Návod k programu

Při spuštění program inicializuje prázdný kořen AVL-stromu. Uživatel dále zadává po jednotlivých řádcích příkazy v podobě klíčového slova, příp. klíčového slova, mezery a parametru.

klíčové slovo	parametr	co takový příkaz dělá
`ins`	*integer*	do stromu přidá uzel se zadanou hodnotou klíče
`del`	*integer*	ze stromu odebere uzel se zadanou hodnotou klíče
`find`	*integer*	vypíše cestu od kořene stromu k uzlu se zadanou hodnotou klíče, resp. uzlu s nejvyšší, resp. nejnižší hodnotou klíče (vypisuje klíče uzlů, kterými procházíme)
	`max`	
	`min`	
`q`		ukončí program

Nedostatky

Program bych chtěl v budoucnu rozšířit o

- možnost vizualizace stromu,
- načítání práce se vstupními daty v podobě textového souboru,
- možností vybrat si mezi běžným binárním stromem, AVL-stromem, červeno-černým stromem, příp. jinými datovými strukturami.

Testovací data

Vstupní data	Výstup programu	Výstup, kdyby šlo o jednoduchý binární vyhledávací strom

`ins 3`
`ins 4`
`ins 5`
`find 4`
	`4`	`3 -> 4`